STRUCTURED PROGRAMMING USING GO

JOSHUA ERDY TAN

REALYN MORENO

Structured Programming using Go

Handbook for Beginning Programmers

Joshua Erdy Tan, LPT, MBDBI

Realyn Moreno

The person or persons who have associated work with this document (the "Dedicator" or "Certifier") hereby either (a) certifies that, to the best of his knowledge, the work of authorship identified is in the public domain of the country from which the work is published, or (b) hereby dedicates whatever copyright the dedicators holds in the work of authorship identified below (the "Work") to the public domain. A certifier, moreover, dedicates any copyright interest he may have in the associated work, and for these purposes, is described as a "dedicator" below.

A certifier has taken reasonable steps to verify the copyright status of this work. Certifier recognizes that his good faith efforts may not shield him from liability if in fact the work certified is not in the public domain.

Dedicator makes this dedication for the benefit of the public at large and to the detriment of the Dedicator's heirs and successors. Dedicator intends this dedication to be an overt act of relinquishment in perpetuity of all present and future rights under copyright law, whether vested or contingent, in the Work. Dedicator understands that such relinquishment of all rights includes the relinquishment of all rights to enforce (by lawsuit or otherwise) those copyrights in the Work.

Dedicator recognizes that, once placed in the public domain, the Work may be freely reproduced, distributed, transmitted, used, modified, built upon, or otherwise exploited by anyone for any purpose, commercial or non-commercial, and in any way, including by methods that have not yet been invented or conceived.

To the extent possible under law, Joshua Erdy Alforja Tan and Realyn Moreno had waived all copyright and related or neighboring rights to Structured Programming using Go. This work is published from the Philippines.

THIS BOOK IS FREE OF CHARGE

TABLE OF CONTENTS

Chapter I:	Introduction to Go Language	1
Chapter II:	Constants in Go Language	11
Chapter III:	Go Language Input/Output	12
	Exercise 1 and Quiz # 1	18
Chapter IV:	Go Arrays	19
Chapter V:	Go Language Conditions and Switch Statement	21
Chapter VI:	Go Language Looping Statements	27
	Exercise 2 and Quiz # 2	28
Chapter VII:	Go Language Functions, Structure and Maps	29
Chapter VIII:	Compiling and Running Go Programs thru OnlineGDB	33
	Exercise 3 and Quiz # 3	34
	APPENDICES	35

CHAPTER I:
INTRODUCTION TO GO LANGUAGE

Introduction to Go

Go is a cross-platform, open source programming language.

Go was developed at Google by Robert Griesemer, Rob Pike, and Ken Thompson in 2007.

Go's syntax is similar to C++.

Go Syntax

A Go file consists of the following:

- Package declaration
- Import packages
- Functions
- Statements and expressions

Example:

```
package main

import ("fmt")

func main() {
  fmt.Println("Hello Philippines!")
}
```

Every program in Go is a component of a package. We use the package keyword to define this. In this instance, the application is a part of the primary package. One can import the files contained in the fmt package using import ("fmt"). Go disregards empty space. Coding is easier to read when there are white spaces. A function denoted is func main(). Any code included within its curly brackets will be carried out.

The fmt package includes a function called fmt.Println(). Text is outputted or printed using it. It will print "Hello Philippines!" in this case.

Go Statements

An example of a Go Statement is fmt.Println("Hello Philippines!"). Statements in Go can be separated by a semicolon (";") or by closing a line (using the Enter key). The line will automatically terminate with ";" when you press the Enter key (does not show up in the source code). There is no place for the left curly bracket at the beginning of a line.

Example:

```
package main; import ("fmt"); func main() { fmt.Println("Hello Philippines!");}
```

Go Comments

A comment is a piece of text that is ignored during execution. Comments can be used to explain and improve the readability of code. When testing alternative code, comments can also be used to prevent code execution. Go allows for single-line or multi-line comments. Single-line comments are preceded by two forward slashes (//). The compiler ignores any text between / and the end of the line (will not be executed). Comments that span multiple lines begin with /* and end with */.

Example (Single Line Comment):

```
// This is a comment
package main
import ("fmt")

func main() {
  // This is a comment
  fmt.Println("Hello Philippines!")
}
```

Example (Multiple Line Comment):

```
package main
import ("fmt")

func main() {
  /* The code below will print Hello Philippines
  to the screen, and it is amazing */
  fmt.Println("Hello Philippines!")
}
```

Go Comments to Prevent Line Execution

You can also use comments to stop the code from running. Save the commented code for future reference and troubleshooting.

Example:

```
package main
import ("fmt")

func main() {
  fmt.Println("Hello Philippines!")
  // fmt.Println("This line does not execute")
}
```

Go Variables and Data Types

Variables are storage containers for data values. Variables in Go can be of various types, such as:

- **int -** holds integers (whole numbers) like 123 or -123.
- **float -** stores decimal floating point numbers such as 19.99 or -19.99.
- **string -** stores text like "Hello World." Double quotes are used to surround string values.
- **bool -** stores values that can be true or false.

Go Boolean Data Types

A Boolean data type is declared with the *bool* keyword and can only take the values true or false. The default value of a Boolean data type is *false*.

Go Boolean Formatting Verbs

The boolean data type is compatible with the following verbs:

Verb	Description
%t	Boolean value
%v	Same as %t

Example:

```
package main
import ("fmt")

func main() {
  var b1 bool = true // typed declaration with initial value
  var b2 = true // untyped declaration with initial value
  var b3 bool // typed declaration without initial value
  b4 := true // untyped declaration with initial value

  fmt.Println(b1) // Returns true
  fmt.Println(b2) // Returns true
  fmt.Println(b3) // Returns false
  fmt.Println(b4) // Returns true
}
```

Go Integer Data Types

The integer data type has two categories:

- *signed* keyword which can take positive and negative number values.
- *unsigned* which can only take positive, non-negative number values.

The default value of an integer data type is *int*.

Go Integer Formatting Verbs

The integer data type is compatible with the following verbs:

Verb	Description
%b	Base 2
%d	Base 10
%+d	Base 10 and always show sign
%o	Base 8
%x, %X	Base 16

Example:

```
package main
import ("fmt")

func main() {
  var x int = 500
  fmt.Println(x) // Returns 500
 }
```

Go Float Data Types

Decimal positive and negative values with a decimal point, such as 35.3, -2.34, or 3597.34987, are stored using the float data types. Float data types in Go had two values, *float32* for 32-bits and *float64* for 64-bits.

The default value of a float data type is float64.

Go Float Formatting Verbs

The float data type is compatible with the following verbs:

Verb	Description
%e	Scientific notation with 'e' as exponent
%f	Decimal point, no exponent
%.2f	Default width, precision 2
%6.2f	Width 6, precision 2
%g	Exponent as needed, only necessary digits

Example:

```
package main
import ("fmt")

func main() {
  var x float64 = 1234.56
  fmt.Printf(x)
}
```

Go String Data Types

A series of characters are stored in the string data type (text). Double quotes must be used to enclose string values.

Go String Formatting Verbs

The string data type is compatible with the following verbs:

Verb	Description
%s	Plain string
%q	Double-quoted string
%x	Hex dump of byte values
% x	Hex dump with spaces

Example:

```
package main
import ("fmt")

func main() {
  var txt string = "Hello!"

  fmt.Printf(txt)
}
```

There are two ways to declare a variable in Go:

1. Using the keyword var: Use the var keyword, followed by the name and type of the variable:
   Syntax: var *variablename type* = *value*
2. With the := sign: Use the := sign, followed by the variable value:
   Syntax: *variablename* := *value*

For better readability, several variable declarations can also be gathered within a block.

Example:

```
package main
import ("fmt")

func main() {
  var student1 string = "Joshua" //type is string
  var student2 = "Realyn" //type is inferred
  x := 100 //type is inferred

  fmt.Println(student1)
  fmt.Println(student2)
  fmt.Println(x)
}
```

Differences using var and :=

Var	**:=**
Useful both inside and outside of functions.	Only applicable within functions.
Variable declaration and value assignment can be accomplished independently.	It is impossible to declare a variable and assign values individually (must be done in the same line).

Examples:

```
package main
import ("fmt")

func main() {
   var (
     a int
     b int = 5
     c string = "mylife"
   )

  fmt.Println(a)
```

```
  fmt.Println(b)
  fmt.Println(c)
}
```

Rules for Go Variable Naming

- A variable name must be preceded by a letter or the underscore symbol ( ).
- The first digit of a variable name cannot be used.
- Only underscores (_, a-z, a-z, 0-9, and a-z) and alphanumeric characters are permitted in variable names.
- Names of variables are case-sensitive (age, Age and AGE are three different variables).
- The variable name can be as long as you want.
- No slashes are permitted in variable names.
- Any Go keywords cannot be used in the variable name.

CHAPTER II:
CONSTANTS IN GO LANGUAGE

Constants in Go

The word "const" designates the variable as "constant," which makes it read-only and immutable. When a constant is declared, its value must be given.

Example:

```
package main
import ("fmt")

const PI = 3.14

func main() {
  fmt.Println(PI)
}
```

CHAPTER III:
GO LANGUAGE
INPUT/OUTPUT

Go User Input

Go offers a variety of options for users to provide input. The following functions can be used to read user inputs:

- Scan()
- Scanln()
- Scanf()

Go Scan() Function

The raw user input is given to the Scan() function as values separated by spaces, and it stores them in the variables. Spaces are added after newlines. There can be multiple user inputs.

Example:

```
package main
import ("fmt")

func main() {
  var i int

  fmt.Print("Type any number: ")
  fmt.Scan(&i)
  fmt.Println("The number is: ", i)
}
```

Go Scanln() Function

Similar to Scan(), Scanln() function stops looking for inputs at a newline (at the press of the Enter key.).

Example:

```
package main
import ("fmt")

func main() {
  var i,j int

  fmt.Print("Type two numbers: ")
  fmt.Scanln(&i, &j)
  fmt.Println("Your numbers are:", i, "and", j)
}
```

Go Scanf() Function

The inputs are received and stored by the Scanf() function according to the formats chosen for its parameters.

Example:

```
package main
import ("fmt")

func main() {
  var i,j int

  fmt.Print("Type two numbers: ")
  fmt.Scanf("%v %v",&i, &j)
  fmt.Println("Your numbers are:", i, "and", j)
}
```

Go Output

Go offers a variety of options for users to provide output. The following functions can be used as outputs:

- Print()
- Println()
- Printf()

Go Print() Function

The default format of the arguments is printed by the Print() function. You can use “\n” to print the arguments in new lines. If you want to add spaces, use “ ”. Print() also inserts spaces if there are no string arguments.

Example 1:

```
package main
import ("fmt")

func main() {
  var i,j string = "Hello","Philippines!"

  fmt.Print(i)
  fmt.Print(j)
}
```

Example 2:

```
package main
import ("fmt")

func main() {
  var i,j string = "Hello","Philippines!"

  fmt.Print(i, "\n")
  fmt.Print(j, "\n" )
}
```

Example 3:

```
package main
import ("fmt")

func main() {
  var i,j string = "Hello","Philippines!"

  fmt.Print(i, " ",j)
}
```

Example 4:

```
package main
import ("fmt")

func main() {
  var i,j = 150,300

  fmt.Print(i,j)
}
```

Go Println() Function

The Println() function is similar to Print() with the addition of a newline at the conclusion and a space between the arguments.

Example:

```
package main
import ("fmt")

func main() {
  var i,j string = "Hello","Philippines!"

  fmt.Println(i,j)
}
```

The Printf() Function

Prior to printing, the Printf() method prepares its arguments using the provided formatting verb. Here, two formatting verbs will be used:

- To print the value of the arguments, use %v.
- To display the type of the arguments, use %T.

Example:

```
package main
import ("fmt")

func main() {
  var i string = “Joshua"
  var j int = 26

  fmt.Printf("i has value: %v and type: %T\n", i, i)
  fmt.Printf("j has value: %v and type: %T", j, j)
}
```

Go Exercise 1

1. Create a Go program that shows both your name and address.
2. Create a Go program and add a comment with your nickname.

Go Quiz # 1

Fill in the blanks with the correct answer.

1. Similar to Scan(), _____ function stops looking for inputs at a newline (at the press of the Enter key.).
2. No _____ are permitted in variable names.
3. The default format of the arguments is printed by the _____ function. You can use “\n” to print the arguments in new lines.
4. _____ is only applicable within functions.
5. A series of characters are stored in the _____ data type (text).

CHAPTER IV:
GO ARRAYS

Go Arrays

Instead of defining distinct variables for each value, arrays are utilized to store numerous values of the same type in a single variable. In Go, there are two ways to construct an array, one by using a var keyword and the other is by with the := sign.

Example 1:

```
package main
import ("fmt")

func main() {
  var arr1 = [4]int{1,2,3,9}
  arr2 := [5]int{4,5,6,7,8}

  fmt.Println(arr1)
  fmt.Println(arr2)
}
```

Example 2:

```
package main
import ("fmt")

func main() {
  var cars = [4]string{"Toyota", "Mitsubishi", "Ford", "Mazda"}
  fmt.Print(cars)
}
```

Accessing Go Arrays

A specific array element can be accessed by using the index number. Array indexes in Go begin at 0. Accordingly, the first element is [0], the second is [1], etc. By using the index number, you can also modify the value of a specific array element.

Example 1:

```
package main
import ("fmt")

func main() {
  prices := [4]int{10,21,30,42}

  fmt.Println(prices[0])
  fmt.Println(prices[2])
}
```

Example 2:

```
package main
import ("fmt")

func main() {
  prices := [4]int{10,20,30,40}

  prices[3] = 12
  fmt.Println(prices)
}
```

CHAPTER V:
GO LANGUAGE CONDITIONS AND SWITCH STATEMENT

Go Conditions

There are four types of conditions in Go:

- if
- if-else
- else-if
- nested if

Go If Statement

Syntax:

```
if condition {
  // code to be executed if condition is true
}
```

Example:

```
package main

import ("fmt")

func main() {
  if 26 > 10 {
    fmt.Println("26 is greater than 10")
  }
}
```

Go If-Else Statements

Syntax:

```
if condition {
  // code to be executed if condition is true
} else {
  // code to be executed if condition is false
}
```

Example:

```
package main

import ("fmt")

func main() {
  time := 22
  if (time < 18) {
    fmt.Println("Good day!")
  } else {
    fmt.Println("Good evening!")
  }
}
```

Go Else-If Statements

Syntax:

```
if condition1 {
  // code to be executed if condition1 is true
} else if condition2 {
  // code to be executed if condition1 is false and condition2 is true
} else {
  // code to be executed if condition1 and condition2 are both false
}
```

Example:

```
package main
import ("fmt")

func main() {
  a := 15
  b := 15
  if a < b {
    fmt.Println("a is less than b.")
  } else if a > b {
    fmt.Println("a is more than b.")
  } else {
    fmt.Println("a and b are equal.")
  }
}
```

Go Nested-If Statements

Syntax:

```
if condition1 {
  // code to be executed if condition1 is true
  if condition2 {
    // code to be executed if both condition1 and condition2 are true
  }
}
```

Example:

```
package main

import ("fmt")

func main() {
  num := 22
  if num >= 15 {
    fmt.Println("Num is more than 15.")
    if num > 20{
      fmt.Println("Num is also more than 15.")
    }
  } else {
    fmt.Println("Num is less than 10.")
  }
}
```

Go Switch Statements

To choose which of several code blocks should be executed, use the *switch* statement.

Go Switch Single-Case Statements

Syntax:

```
switch expression {
case x:
  // code block
case y:
  // code block
case z:
...
default:
  // code block
}
```

Example:

```
package main
import ("fmt")
func main() {
 food := 3
 switch food {
 case 1:
  fmt.Println("Pizza")
 case 2:
  fmt.Println("Burger")
 case 3:
  fmt.Println("Chicken")
  }
}
```

Go Switch Multiple-Case Statements

Syntax:

```
switch expression {
case x,y:
  // code block
case v,w:
  // code block
case z:
...
default:
  // code block
}
```

Example:

```
package main

import ("fmt")

func main() {
  day := 5

  switch day {
  case 1,3,5:
   fmt.Println("Odd weekday")
  case 2,4:
    fmt.Println("Even weekday")
  case 6,7:
   fmt.Println("Weekend")
 default:
   fmt.Println("Invalid day of day number")
 }
```

CHAPTER VI:
GO LANGUAGE
LOOPING STATEMENTS

Go Loops

The *for* loop iterates over a code block a predetermined number of times. In Go, there is only one loop, the for loop.

Syntax:

```
for statement1; statement2; statement3 {
  // code to be executed for each iteration
}
```

Example:

```
package main

import ("fmt")

func main() {
  for i:=0; i < 10; i++ {
    fmt.Println(i)
  }
}
```

Go Exercise 2

1. Create a Go Program with multiple strings.
2. Create a Go program which holds three names as string variables.
3. Make a Go program that lets a user input the following details:
   a. Name
   b. Age
   c. Favorite Snack

Go Quiz # 2

Fill in the blanks with the correct answer.

1. In Go, there is only one loop, the _____ loop.
2. To choose which of several code blocks should be executed, use the _____ statement.
3. A specific _____ element can be accessed by using the index number.
4. There are _____ types of conditions in Go.
5. In Go, there are two ways to construct an array, one by using a _____ keyword and the other is by with the := sign

CHAPTER VII:
GO LANGUAGE FUNCTIONS, STRUCTURE AND MAPS

Go Function

A block of statements called a *function* can be utilized repeatedly throughout a program. A function won't start running immediately after a page loads. A call to a function will cause it to be used. To create a function in Go, use the *func* keyword.

Syntax:

```
func FunctionName() {
  // code to be executed
}
```

Example:

```
package main

import ("fmt")

func myCall() {
  fmt.Println("I just love you!")
}

func main() {
  myCall() // call the function
}
```

Go Struct

A collection of members of various data types are assembled into a single variable using a *struct*, which is short for structure. Structs are used to store many values of various data kinds in a single variable as opposed to arrays, which are used to store multiple values of the same data type. For the purpose of creating records, a struct can be helpful.

Syntax:

```
type struct_name struct {
  member1 datatype;
  member2 datatype;
  member3 datatype;
  ...
}
```

Example:

```
package main

import ("fmt")

type Person struct {
  name string
  age int
  job string
  salary int
}

func main() {
  var persl Person

  // Persl specification
  persl.name = "Josh"
```

```
  pers1.age = 27
  pers1.job = "Professor"
  pers1.salary = 15000

  // Access and print Pers1 info
  fmt.Println("Name: ", pers1.name)
  fmt.Println("Age: ", pers1.age)
  fmt.Println("Job: ", pers1.job)
  fmt.Println("Salary: ", pers1.salary)
}
```

Go Maps

Key:value pairs are used to hold data values in maps. A map's elements are all key:value pairs. A map is an unsorted, flexible collection that forbids duplications. The quantity of a map's elements determines its length. The *len()* method can be used to locate it. A map's default value is zero. References to an underlying hash table are stored in maps. Maps can be made in Go in a variety of ways.

Syntax:

var *a* = map[KeyType]ValueType{*key1*:*value1*, *key2*:*value2*,...}
b := map[KeyType]ValueType{*key1*:*value1*, *key2*:*value2*,...}

Example:

```
package main

import ("fmt")

func main() {
  var a = map[string]string{"brand": "Ford", "model": "Mustang", "year": "1997"}
  b := map[string]int{"Oslo": 1, "Manila": 2, "Pyongyang": 3, "Beijing": 4}
```

```
  fmt.Printf("a\t%v\n", a)
  fmt.Printf("b\t%v\n", b)
}
```

Create Maps using make() Function

You can create maps using make() function.

Syntax:

var *a* = make(map[KeyType]ValueType)
b := make(map[KeyType]ValueType)

Example:

```
package main

import ("fmt")

func main() {
  var a = make(map[string]string) // The map is empty now
  a["brand"] = "Ford"
  a["model"] = "Mustang"
  a["year"] = "1964"
                        // a is no longer empty
  b := make(map[string]int)
  b["Oslo"] = 1
  b["Manila"] = 2
  b["Pyongyang"] = 3
  b["Beijing"] = 4

  fmt.Printf("a\t%v\n", a)
  fmt.Printf("b\t%v\n", b)
}
```

CHAPTER VIII: COMPILING AND RUNNING GO PROGRAMS THRU ONLINEGDB

Compiling and Running Go Language Programs thru OnlineGDB

You can compile and run Go Language Programs thru OnlineGDB:

1. First, open onlinegdb.com.
2. Second, choose Go on the Language option on the upper-right side of the website.
3. Third, start coding!
4. Finally, run and test your code!

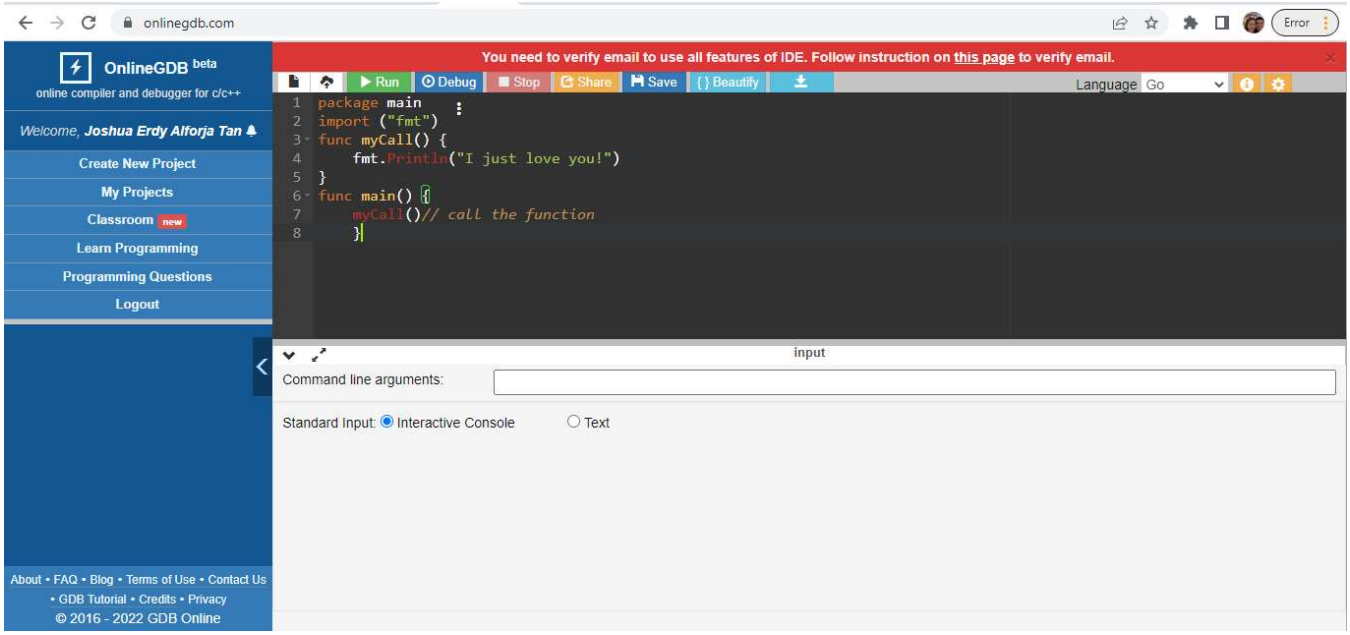

Figure 1. The OnlineGDB interface.

Go Exercise 3

1. Create a Go program that calculates the average grades of the following: 85, 88, 92, 100, 99.
2. Create an Array from numbers 1 to 5.

Go Quiz # 3

Fill in the blanks with the correct answer.

1. You can create maps using the _____ function.
2. A collection of members of various data types are assembled into a single variable using a _____.
3. A block of statements called a _____ can be utilized repeatedly throughout a program.
4. _____ are used to hold data values in maps.
5. To create a function in Go, use the _____ keyword.

APPENDICES

APPENDIX I: QUIZ ANSWERS

QUIZ # 1

1. Scanln()
2. slashes
3. Print()
4. :=
5. string

QUIZ # 2

1. for
2. switch
3. array
4. four, 4
5. var

QUIZ # 3

1. make()
2. struct
3. function
4. Key:value pairs
5. func

APPENDIX II: REFERENCES

OnlineGDB.com (2022). *OnlineGDB Compiler and Editor*. Retrieved August 15, 2022 from https://www.onlinegdb.com

Wikimedia Foundation. (2022, August 13). *Go (programming language)*. Wikipedia. Retrieved August 15, 2022, from https://en.wikipedia.org/wiki/Go_(programming_language)

W3Schools.com. (2022). *Go Tutorial*. Retrieved August 15, 2022, from https://www.w3schools.com/go/index.php

ABOUT THE AUTHORS

JOSHUA ERDY TAN, LPT, MBDBI is currently a freelance programmer and Data Scientist. He graduated with a Bachelor of Science in Information and Communications Technology Education degree from the Philippine Normal University-Manila in 2015 and a Master in Big Data and Business Intelligence from European Business School of Barcelona (ENEB) and Universidad Isabel I de Castilla. He taught subjects such as Computer Programming I, Quantitative Methods, and Emerging Technologies and Applications Development at Saint Vincent de Ferrer College of Camarin, Inc. in Camarin, Caloocan City. Besides of having a vast majority of knowledge in Linux, Microsoft Office, and some Computer Programming languages, he also worked in several Local Government Units (LGUs) in Makati City, particularly in Barangay Urdaneta and Post Proper Northside.

REALYN MORENO is currently a GSO Junior Staff/Secretary at Mary Johnston Hospital in Manila. She graduated with an Information Technology diploma from Microlink Institute of Science and Technology in Bulacan 2017. Prior to being a GSO Junior Staff, she also worked in Mary Johnston Hospital as an Administrative Secretary from September 2018 to April 2022 and as a Barangay Secretary in Binagbag, Angat, Bulacan.